Review on MSD Radix Sort

Implementation and Improvement Study

Wangzi Wen & Zuyue Xie

Abstract

We implemented MSD (most significant digit) radix sort to compete with other sorting algorithms to sort random Chinese characters. In the early study of radix sort, we expect it to be the fastest to sort Chinese character. As one of non-comparison sort, MSD radix sort has advantages compare with other sorting algorithms by handling string of variable length and it can order a string prior to scan the entire of the string. However, unlike English word can be order by English alphabet, to order Chinese character is quite complex. As we investigated both sorting Chinese characters (Pinyin) in English alphabet order and Chinese character in Pinyin order. We demonstrated three alternative implementation of MSD radix sort to sort Chinese character as we eager to find the most comparative one in terms of time complexity among other sorting algorithms.

1. INTRODUCTION

With the improvement of technology over the years, words and sentences become just little bits that fly in the air. Those bits of data may seem no harm to anything, yet as the raising demand of data exchanging, the huge amount of data can be generated in flash second that need to be properly taking care and maintaining in its maximum performance. A sorting algorithm which puts elements of data into some order is the way of resolve the issue.

In our study of MSD radix sort, we have knowledges of on the high efficiency of those kind of sorting algorithm when sorting strings. In the ideal case of string value, MSD radix sort can produce a time complexity that close to the big O of (N + R). There are two key point of optimization that we eager to approach on MSD radix to allow it sort Chinese character efficiently. First, since MSD radix sort starts counting the elements of a value from its left most digit to the right, and the operation of checking the next element was done recursively based on the single value length, once those significant digits are counted the rest of digits does not matter in terms of ordering if those significant digits are distinguishable or unique. Second, as the implementation of our MSD radix sort was done in java, and java have essentially two data types, primitive and reference types, counting a reference data type might some have differences when counting primitive data type.

By carrying those two key points in mind, we start to search and implement our alternative MSD radix sort for Chinese character.

2. THE PERFORMANCE OF FASTBIT – RADIX SORT

The following study is a research which previously done to improve integer value radix sort by converting integer to bit value and sort it by counting bits.

(S & Arya, 2016) The dataset for sorting consists of N positive integer data to be sorted. In Fastbit Radix-sort, each positive integer data are represented with set of bits of size B, which can be 32 or 64 depending on the machine architecture. Each pass in Fastbit Radix-sort is based on sorting of binary string which is having best case as (N/2) and worst case as N.

(S & Arya, 2016) From the analysis statistics we can derive observation that though the fastbit radix sort has more number of steps counts than the traditional radix sort, in terms of execution time the fastbit radix sort is significantly 40% more faster than the traditional radix sort.

As this study suggested, radix sort have better performance when sort with bit that integer values. Even when the longer bit value converted from integer value. Assume if each Chinese character just like English can be sort in alphabet order, there is a Chinese alphabet table, Discarding the length of that table. It is possible that converting Chinese character from string to other primitive data type in java and sort it will be faster than just sort the string value.

3. THE HUSKY ENCODING

The following study show that encode can be done in order to minimize inversions and improve the sorting process.

(Hillyard et al., 2020) It is not essential that coding is perfect for Huskysort to work well. We have experimented with deliberately throwing off the encoding quite drastically. If the probability of one of these deliberate mis-codings is less than approximately 25 percent, Husky sort can still save time.

4. Discussion the order of Chinese characters

In our study of how to order Chinese characters, we mainly focused on the Pinyin order. Pinyin is the romanization of Chinese character based on their pronunciation, and this is the reason why unlike English words each letter will influent the order based on the alphabet. Depending about the sound of Chinese characters combination each letter of Pinyin may not correspond the correct order. Therefore, comparing or counting the letters of Pinyin will not sorting Chinese character correctly in lot of cases, and with the idea of fastbit radix sort in mind, there is another way of sorting Chinese characters without convert it from string to Pinyin string.

5. Discussion on encoding Chinese characters

During our research we discovered the library called Collator, and what it does is translate Chinese character to a unique CollationKey. The CollationKey them can be convert to a byte array that correspond that CollationKey. Form our own perspective after series of testing, each two bytes on that byte array may be represents the single Chinese character's Pinyin order. This operation is quite like the encoding part of husky sort, the only difference is that the length of byte array which generate from CollationKey is changing based on the length of Chinese character combinations. Since after using Collator, we avoid counting or compare each letter of Pinyin, it is possible to sort Chinese character in a correct order.

6. Conclusion and Optimization

Combining with the research that previously done as well as the discussion on Chinese character. There are three ways to optimized MSD radix sort on Chinese.

6.1 Byte Node implementation

Storing both Chinese characters and a byte array from CollationKey in a node. When counting take place in MSD radix, it will access and count the byte array of each node. With this implementation, event conversion taken place, the overall performance is still fast as it is counting on primitive data type.

6.2 2-D byte array with fixed length implementation

CollationKey byte	0s	UTF-8 bytes

As the graph above, first encoding CollationKey of Chinese characters to byte array as well as UTF-8 format byte array. By having an outer length of number of Chinese characters and inner array of length

21 filled with 0 bytes. Since the maximum length of byte array convert from CollationKey on a 3-character Chinese name is 11, this case we address and copy the index from 0 to 12 to be as CollationKey Byte array and the rest is UTF-8 byte array. After MSD sort, we convert back the Chinese character from the UTF-8 format bytes from the last part of this array. However, a vital issue for this approach was it will fail when the Chinese characters is less or large than 3 since we are only addressing a fixed length of Collation key on the array and stop MSD recursion to look next bytes when it seems 12 digits already, yet there may have extra 0 bytes in the first 12 element and it will influence the ordering.

6.3 2-D byte array implementation

CollationKey bytes	UTF-8 bytes

Similar to the 2-D byte array with fixed length, encoding Chinese Characters to both CollationKey byte array and UTF-8 byte array and copy them into one byte array. This approach indeed solves the issue where previous approach has, it can sort Chinese character in variable length, since MSD sort is looking for byte from left to right, when it is reaching the last byte of CollationKey byte even it will still take more time recursive through the rest of bytes, the correct order is generated. However, an issue with this approach is when covert the UTF-8 bytes back to Chinese Characters, since the length of this array is unknow, it will need separate cases to locate the UTF-8 bytes on that array.

By and large, it is possible to sort Chinese character on MSD radix with better time complexity, as (S & Arya, 2016) suggest convert the source to a different data type and merging some idea from husky sort (Hillyard et al., 2020). The final solution for our MSD radix sort with 2-d byte array implementation would be trying to get a fixed length of UTF-8 byte array, since we know where to look the UTF-8 byte after sorting complete, there will be no separate function needed to convert it back to Chinese characters.

7. Reference

S, A. V. K., & Arya, A. (2016). Fastbit-radix sort: Optimized version of radix sort. 2016 11th International Conference on Computer Engineering Systems (ICCES), 305–312. https://doi.org/10.1109/ICCES.2016.7822019

Hillyard, R. C., Liaozheng, Y., & R, S. V. K. (2020). Huskysort.